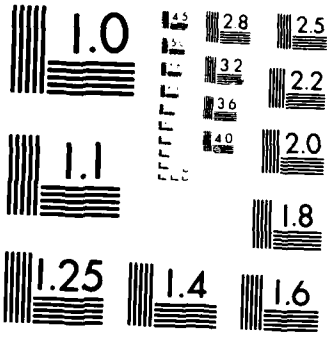
AD-A152 174 MULTIPLE ARRAY PROCESSORS FOR OCEAN ACOUSTIC PROBLEMS (U) YALE UNIV NEW HAVEN CT DEPT OF COMPUTER SCIENCE M H SCHULTZ FEB 85 YALEU/DCS/RR-363 N00014-82-K-0184 1/1

UNCLASSIFIED F/G 20/1 NL

END



(2)

Multiple Array Processors for Ocean Acoustic Problems

Martin H. Schultz

Research Report YALEU/DCS/RR-363
February 1985

YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Abstract. The current revolution in computer hardware and architecture is having a major impact on many disciplines. In this paper we examine some of the implications of this revolution for some problems in computational underwater acoustics. In particular we consider the application of some model multi-array processor systems to the numerical solution of the three-dimensional parabolic approximation, [2, 3] for a discussion of parabolic approximations in underwater acoustics.

Our goal is to illuminate some of the issues involved in taking advantage of the current advances in hardware technology to produce an extraordinarily fast, inexpensive, and portable computer system for simulating underwater acoustic wave propagation.

Multiple Array Processors for Ocean Acoustic Problems

Martin H. Schultz

Research Report YALEU/DCS/RR-363
February 1985

This work was supported in part by ONR grant N00014-82-K-0184 and in part by a joint study with IBM/Kingston.

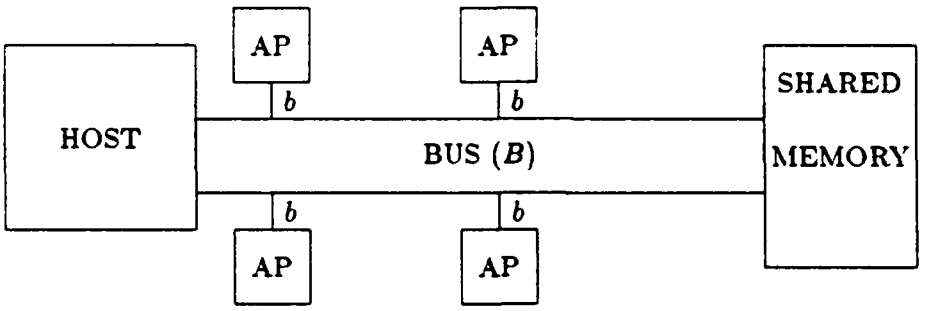
1. Introduction

The current revolution in computer hardware and architecture is having a major impact on many disciplines. In this paper we examine some of the implications of this revolution for some problems in computational underwater acoustics. In particular we consider the application of some model multi–array processor systems to the numerical solution of the three–dimensional parabolic approximation, [2, 3] for a discussion of parabolic approximations in underwater acoustics.

Our goal is to illuminate some of the issues involved in taking advantage of the current advances in hardware technology to produce an extraordinarily fast, inexpensive, and portable computer system for simulating underwater acoustic wave propagation. We refer to the work of Fred Tappert [4] for a discussion of some of the issues involved in using a single array processor system, which he calls PESOGEN (Parablic Equation Solution GENerator), for these problems.

A-1

2. A Brief Overview of Architecture



A typical multi-array processor system would look something like above sketch where the bus has a total bandwidth of B bytes/second, AP represents an array processor which can communicate with the bus at b bytes/second and compute at s multiply/add pairs per second. In Table 1 we present a list of contemporary array processors, their dates of introduction, their megaflop ratings, and their approximate costs.

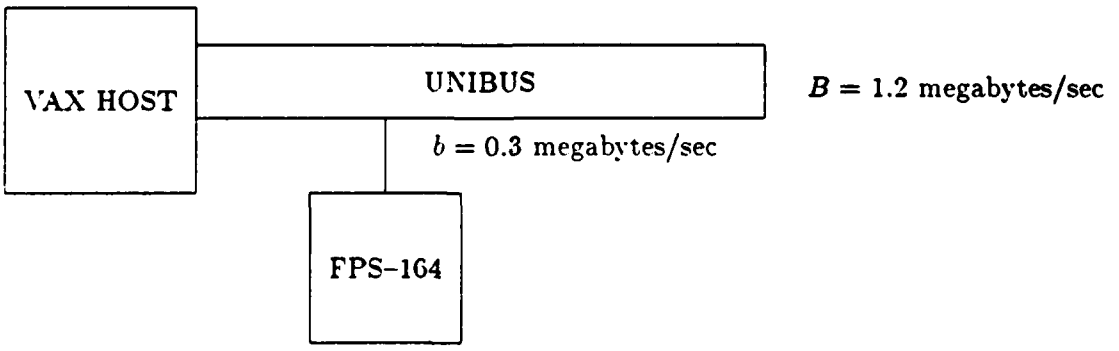
YEAR	NAME	MEGAFLOPS	APPROXIMATE COST (IN THOUSANDS OF DOLLARS)
1982	FPS-164	11	300
1984	MARS-432	30	100
1985	ZIP-3232	20	20

Table 1

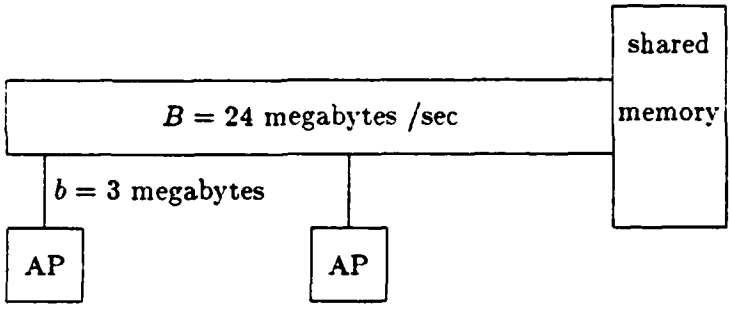
From Table 1 we can see a clear trend of more power for less money (also in a smaller package).

We consider a number of contemporary interconnect schemes :

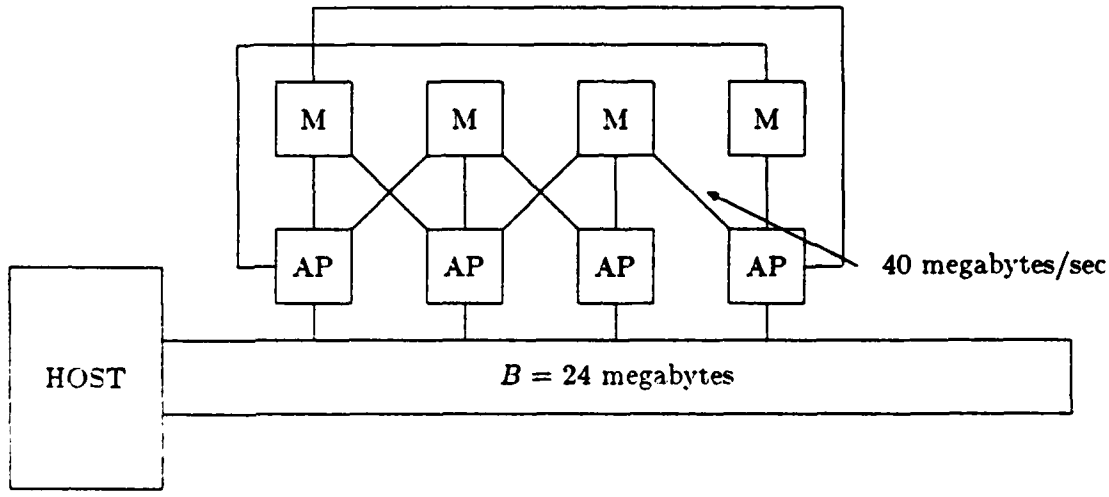
(1) VAX:



(2) APTEC DPS-2400 (a fast UNIBUS):

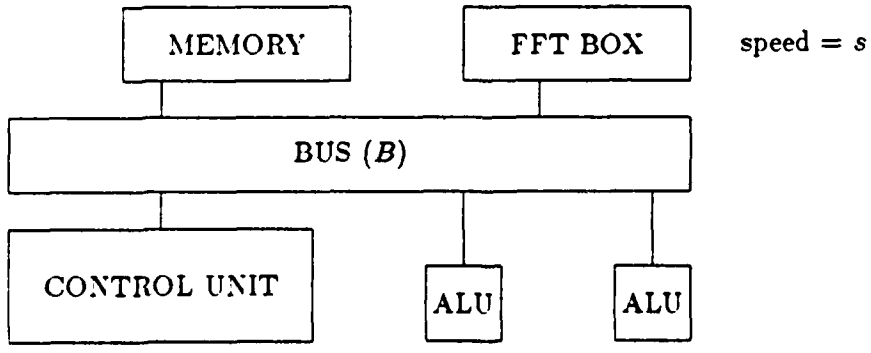


(3) YALE SHARED-BULK-MEMORY (ELI CIRCUS):



This yields an effective bandwidth of 20 megabytes/sec between every pair of adjacent array processors *in parallel.* This system has been implemented at Yale with a single FPS-164 and a single (bulk) memory containing 32 megabytes and expandable to 796 megabytes. It is being implemented by Enrico Clementi at IBM with an IBM mainframe host and at least 10 FPS-164s.

It is currently very fashionable to talk about having special purpose devices integrated into computer systems. In particular, we know how to build very fast FFT devices which could be



integrated into a typical array processor as above (ALU stands for arithmetic–logic unit). Suppose we wish to do complex FFTs based on $n = 1024$ points. We assume that, the data resides in memory, must be moved to the FFT box and then moved back to memory after being transformed. Furthermore, there is little point to building the FFT box unless this process is compute bound (as distinct from I/O or data movement bound), i.e.,

$$\frac{16n}{B} \leq \frac{4n \log n}{s}$$

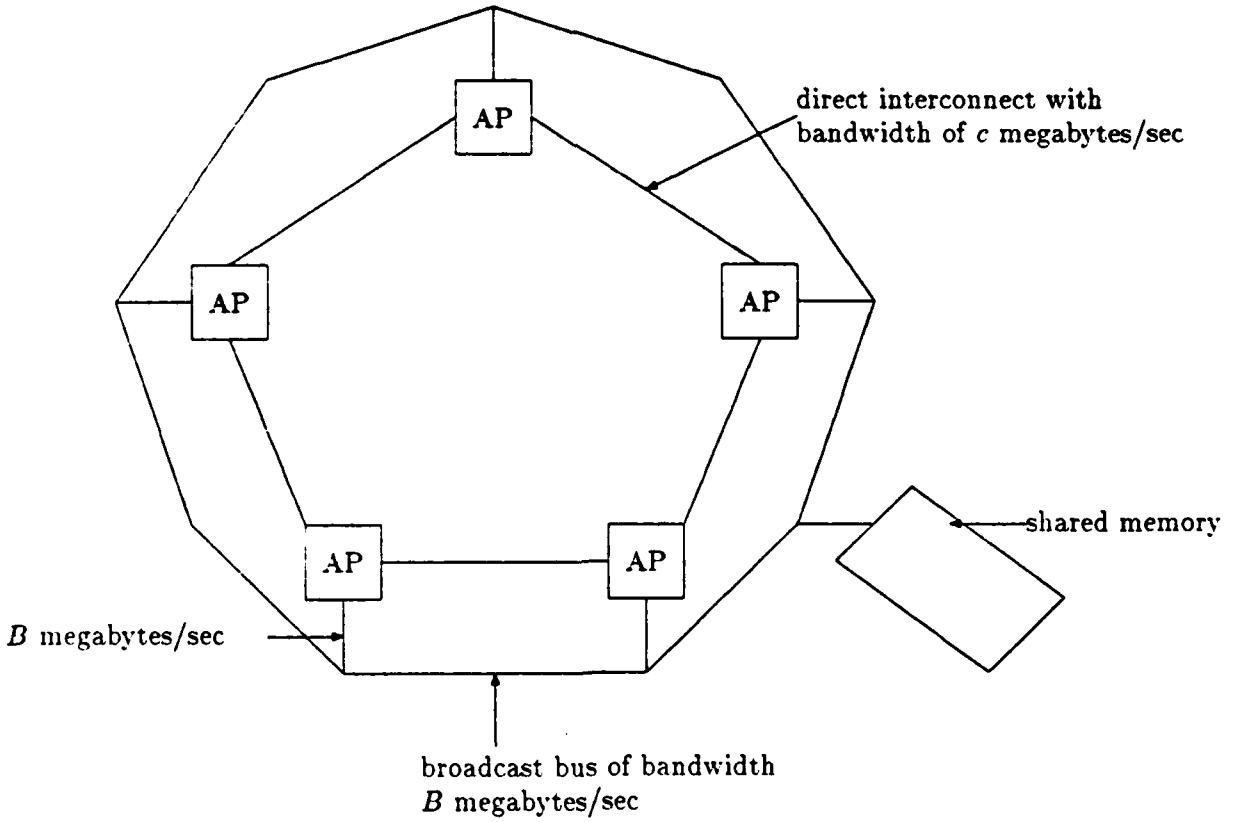
data movement
(4 byte complex arithmetic)

computation

$$\text{or} \quad \frac{s}{B} \leq \frac{1}{4} \log n = 2.5 \quad \text{or} \quad s \leq 2.5 \times B.$$

This inequality tells us that for this situation there is little point in making the FFT box arbitrarily fast. In fact, an upper bound for its speed is provided in terms of the bus bandwidth.

Our general approach is based on what we call the *ELI CIRCUS ALGORITHM DESIGN DISCIPLINE* , i.e., we think of a model architecture that generalizes the one based on the



Yale Shared–Bulk–Memory as shown in the sketch above. What is important for the inventor and implementor of parallel algorithms is the overall simple structure of this target architecture. This

structure is sufficiently simple to allow the formulation and analysis of parallel algorithms. The exact details of the hardware implementation are completely unimportant at this level. Indeed we envision having an automatic system for mapping algorithms for the *ELI CIRCUS* onto specific implementations.

3. A Computation Intensive Ocean Acoustics Problem

We consider the problem of modelling the propagation of a single frequency underwater acoustic signal by means of a 3–dimensional parabolic approximation [2] such an approximation corresponds to the equation

$$u_r = \frac{i}{2}k_0(n^2-1)u + \frac{i}{2k_0}u_{zz} + \frac{i}{2k_0}\frac{1}{r^2}u_{\theta\theta}, \tag{1}$$

[2, 3]. Algorithms for solving Eq. (1) are known to be very computation intensive so that the application of a multi–array processor is quite appropriate.

We consider two basic algorithms for computing the solution to Eq. (1) :

1. The split step Fourier method due to Fred Tappert, [4], which involves two 2–dimensional FFTs per range step; and
2. The explicit finite difference approximations due to Tony Chan, Ding Lee, and Long–jun Shen, [1], which involves a matrix–vector product with a 5–diagonal matrix per range step.

We now describe implementations and analyze the performance of these two algorithms on an *ELI CIRCUS* architecture.

4. The Split Step Fourier Method.

Since most of the work for the split step algorithm is in the 2–dimensional FFTs, we limit our discussion to that. In particular, we consider a 2–dimensional FFT on a $2^n \times 2^n$ mesh using a k processor *ELI CIRCUS* machine. The general idea is to use an algorithm of the following form.

2D–FFT ALGORITHM FOR CIRCUS (k) :

1. Partition the mesh into k equal vertical pieces (assuming k divides 2^n) ;
2. Map each piece into an AP;
3. Do the 1–dimensional FFTs in each AP;
4. Map the transformed data back into shared memory;
5. Partition the mesh into k equal horizontal pieces (transpose done in hardware);
6. Map each piece into an AP;
7. Do the 1–dimensional FFTs in each AP;
8. Map the transformed data back into shared memory.

To apply this type of approach to the split step Fourier algorithm we take $N \equiv 2^n$ points in both the z (depth) and θ (angle) variables and assume the number of processors, k, divides N. In each of steps 3 and 7 of the above algorithm, each processor does

(1) $\frac{N}{k}$ 1–dimensional FFTs on N points which requires

(2) $$\frac{N}{k}\frac{4N\log N}{s} \quad \text{seconds.}$$

We can accomplish steps 4–6 by a variety of means. For example, by having each processor use the broadcast bus to broadcast its transformed data to all the others which would take

(3) $$kT + \frac{8\cdot\frac{N}{k}\cdot N}{b}\cdot k = kT + \frac{8N^2}{b} \quad \text{seconds,}$$

where T is the data transfer startup time or latency. Summing two times the bound in (2) and (3) we get a total time of

(4) $$kT + \frac{8N^2}{b} + \frac{8N^2\log N}{ks} \quad \text{seconds,}$$

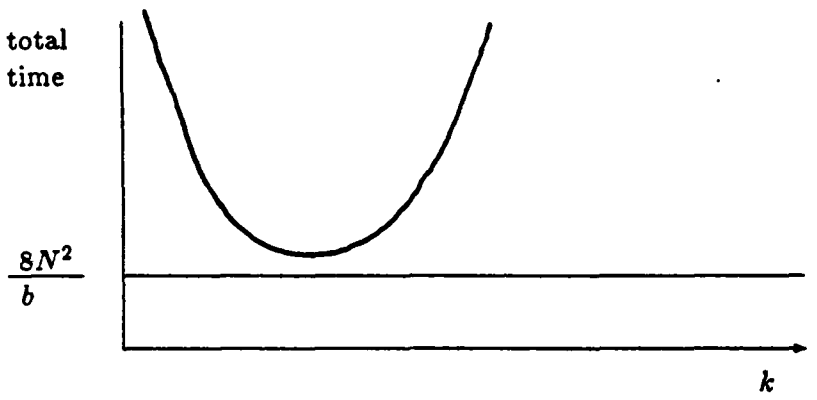


Figure 1

which is graphed in Figure 1 as a function of k. If $B > b$, we can use the shared memory to more effectively move the data. In fact, steps 4–6 would require

(5) $$kT + \frac{16N^2}{B} \quad \text{seconds.}$$

This yields a total time of

(6) $$kT + \frac{16N^2}{B} + \frac{8N^2\log N}{ks} \quad \text{seconds,}$$

which is less than (4) if $2b \leq B$.

Still another way of accomplishing the data movement in steps 4–6 is to use the direct nearest neighbor connections as follows. We base this discussion on the model with the local shared memory previously described, i.e., the Yale Shared–Bulk–Memory.

PIPELINE DATA MOVEMENT ALGORITHM (k) :

For $j = k-1$ **to** 1 **step** -1 :

1. Each AP writes $\frac{N^2}{k}\left(\frac{j}{k}\right)$ data to its right shared memory

2. Each AP reads the data from its left shared memory

Using this approach the total time is given by

$$2Tk + \frac{1}{2}\frac{N^2}{c} + \frac{8N^2 \log N}{ks} \quad \text{seconds.} \tag{7}$$

yielding a graph which is basically similar to that displayed in Figure 1.

It is clear from Figure 1 that as we add more nodes the running time will eventually increase and may eventually exceed the running time with one node.

Let's examine some typical cases for $N = 1024$.

1. VAX and FPS–164s : $T = 3 \times 10^{-3}$, $b = 0.3 \times 10^6$, $s = 10 \times 10^6$

$$\text{TOTAL TIME} = 3000k + \frac{8 \times 10^6}{0.3 \times 10^6} + \frac{8 \times 10^6 \times 10}{10 \times 10^6 \times k} \quad \text{seconds}$$
$$\approx 24 + \frac{8}{k} \quad \text{seconds.}$$

Thus for $k = 2$; TOTAL TIME ≈ 28 seconds
$k = 3$; TOTAL TIME ≈ 26.3 seconds
$k = \infty$; TOTAL TIME ≈ 24 seconds

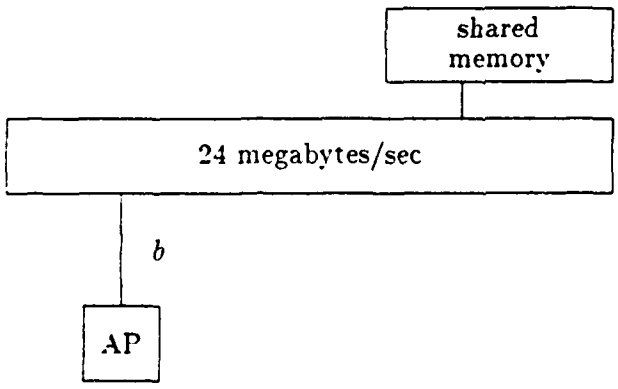
However, we remark that for a single processor system the TOTAL TIME = 8 seconds because there is no data movement.

2. APTEC BUS and FPS–164s : $T = 3 \times 10^{-3}$, $b = 3 \times 10^6$, $s = 10 \times 10^6$

$$\text{TOTAL TIME} \approx 2.4 + \frac{8}{k} \quad \text{seconds}$$

Thus for $k = 2$; TOTAL TIME ≈ 6.4 seconds
$k = 4$; TOTAL TIME ≈ 4.4 seconds
$k = \infty$; TOTAL TIME ≈ 2.4 seconds

There is an interesting question about the use of the shared memory on the APTEC BUS. Suppose b can have any value between 3 and 12 megabytes/sec. The question is how hard we should try



to make b large. If $k \cdot b \geq 24$, e.g., if $k \geq 8$ then using the shared memory approach we have from equation (6) that

$$\text{TOTAL TIME (MEMORY)} \approx \frac{16}{24} + \frac{8}{k} \quad \text{seconds,}$$

while using only the broadcast bus we have from equation (4) that

$$\text{TOTAL TIME (BUS)} \approx \frac{8}{b} + \frac{8}{k} \quad \text{seconds.}$$

Thus if $k \cdot b \geq 24$, we should use the shared memory approach and there is no advantage in making $b > 3$.

5. An Explicit Finite Difference Scheme

We now switch to discussing an explicit 5-point finite difference approximations to the parabolic equation. If we block the unknowns by lines then we can step the solution out in range by using the explicit scheme

$$u^{n+1} = Au^n + Du^{n-1} + g^n$$

where A and D are N-block tridiagonal (with an additional two off diagonal blocks because of the periodicity in the θ variable). If we partition the data into k equal vertical slices each of which is assigned to a processor then for each range step each processor must send its left-most vertical line of data to its left-hand neighbor and its right-most vertical line of data to its right-hand neighbor. If we number the processors, we can envision this algorithm for data movement as follows. In sequence, each odd numbered processor first broadcasts its left-most vertical line of data and then its right-most vertical line of data. Then the even numbered processors do the same. This requires data movement time equal to

$$2 \cdot \left(2 \cdot \frac{8N}{b} \cdot \frac{k}{2} + \frac{k}{2} \cdot T \right) = kT + \frac{16Nk}{b} \quad \text{seconds.} \tag{8}$$

However, the compute time (for the N-block tridiagonal part) is given by

$$\frac{20N^2}{sk} \quad \text{seconds,} \tag{9}$$

yielding a total time of

$$kT + \frac{16Nk}{b} + \frac{20N^2}{sk} \quad \text{seconds.} \tag{10}$$

The graph of this expression (10) as a function of k is similar to the graph of Figure 1. However, the asymptotic behavior of (10) as k increases is significantly worse than that of (3) because in (10) k appears in the numerator of the data transfer term as well as the startups term while in (3) it appears in the numerator of only the startups term. Moreover, the data transfer term is likely to be more important than the startups term.

However, now the story is quite different from the split step algorithm. When we do the data movement of the previous algorithm using the nearest neighbor connection links instead of the broadcast bus, we can use all the links in parallel and hence the time to move the data is given by

$$2T + \frac{16N}{c} \quad \text{seconds,} \tag{11}$$

and the runing time is given by

$$2T + \frac{16N}{c} + \frac{20N^2}{sk} \quad \text{seconds.} \tag{12}$$

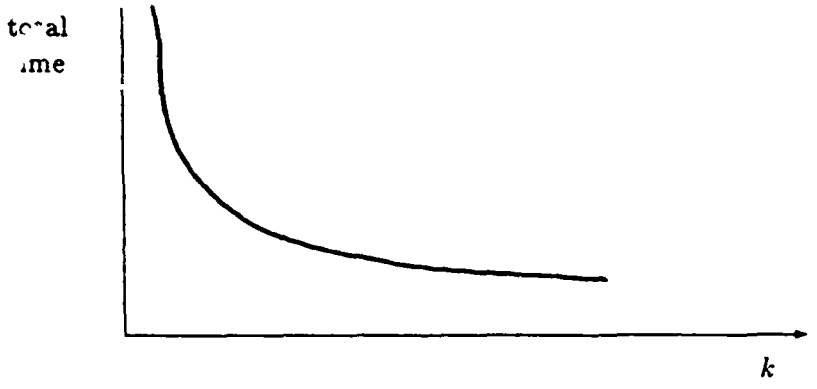


Figure 2

which is pictured in Figure 2 and is monotonically decreasing in k. As before we give some explicit examples with $N = 1024$.

1. VAX with FPS–164s : $T = 3 \times 10^{-3}$, $b = 0.3 \times 10^6$, $s = 10 \times 10^6$ (using the bus).

$$\text{TOTAL TIME} = \frac{16000k}{0.3 \times 10^6} + \frac{3000k}{10^3} + \frac{20 \times 10^6}{10 \times 10^6 \times k} \quad \text{seconds.}$$
$$\approx 24k + \frac{2}{k} \quad \text{seconds.}$$

Hence, $k_{opt} \approx 6$ and $(\text{TOTAL TIME})_{opt} \approx 0.6$ seconds.

2. APTEC BUS and FPS–164s : $T = 3 \times 10^{-3}$, $b = 3 \times 10^6$, $s = 10 \times 10^6$ (using the bus).

Now $k_{opt} \approx 30$ and $(\text{TOTAL TIME})_{opt} \approx 0.15$ seconds.

3. ELI CIRCUS using nearest neighbor direct links : $c = 20 \times 10^6$

$$\text{TOTAL TIME} \approx \frac{16 \times 1000}{20} + \frac{20 \times 10^6}{10 \times 10^6 \times k} \quad \text{seconds}$$
$$\approx \frac{2}{k} \quad \text{seconds.}$$

Hence, $k = 100$ implies a TOTAL TIME ≈ 0.02 seconds.

6. Summary

The solution of the 3–dimensional parabolic approximation is computation intensive. The algorithm of choice on a multi–processor system is very dependent on the system architecture. For bus–based architectures the split step Fourier method is likely to be superior to explicit finite difference methods while on a ring–based machine with independent nearest neighbor interconnects, the opposite is probably true.

Furthermore, for reasonable system parameters we are likely to see almost linear speedups for small to moderate number of processors. However, for bus–based architectures the running time will eventually start to increase as the number of processors is increased.

References

[1] T.F. Chan, Ding Lee, Longjun Shen. *Stable Explicit Schemes for Equations of the Schrödinger Type*. Technical Report YALEU/DC/TR-305, Dept. of Computer Science, Yale Univ., (1984).

[2] Ding Lee. *The State-of-the Art Parabolic Equation Approximation as applied to Underwater Acoustic Propagation with Discussions on Intensive Computations.*, Technical Report NUSC/7247, NUSC, (1984).

[3] F.D. Tappert, *The parabolic equation approximation method*, Lecture Notes in Physics, Springer-Verlag, Heidelberg, 70 (1977).

[4] ————, *Analysis of the Split-Step Algorithm for Parabolic Wave Equation*, These Proceedings, (1985).

END

FILMED

5-85

DTIC